1、Ajax特点

优点：

- 可以无需刷新页面与服务器进行通信
- 允许根据用户事件来跟新部分页面内容

缺点：

- 没有浏览记录，不能回退
- 存在跨域问题
- SEO不友好（爬虫获取不到Ajax请求的数据）

get与post的区别：

1. get请求一般是去取获取数据(其实也可以提交,但常见的是获取数据); post请求一般是去提交数据。
2. get因为参数会放在url中, 所以隐私性, 安全性较差, 请求的数据长度是有限制的, 不同的浏览器和服务器不同, 一般限制在 2~8K 之间, 更加常见的是 1k 以内; post请求是没有的长度限制, 请求数据是放在body中;
3. get请求刷新服务器或者回退没有影响, post请求回退时会重新提交数据请求。
4. get请求可以被缓存, post请求不会被缓存。

2、HTTP协议

1、请求报文

1、请求行

类型：get post

URL路径

HTTP协议版本

2、请求头

格式：

- Host：baidu.com
- Cookie：name=guang
- Content-type：application/x-www-form-urllencoded
- User-Agent：chrome 83

3、请求空行

必须要有

4、请求体

- get请求：请求体为空
- post请求：可为空可不为空，eg：username=guang&password=123456

2、响应报文

1. 响应行

- HTTP协议版本
- 响应状态码：200、404、403、500、401等
- 响应状态字符串：和状态码相同

2. 响应头

格式：

- Content-Type: text/html; charset=utf8
- Content-length: 2048
- Content-encoding: gzip

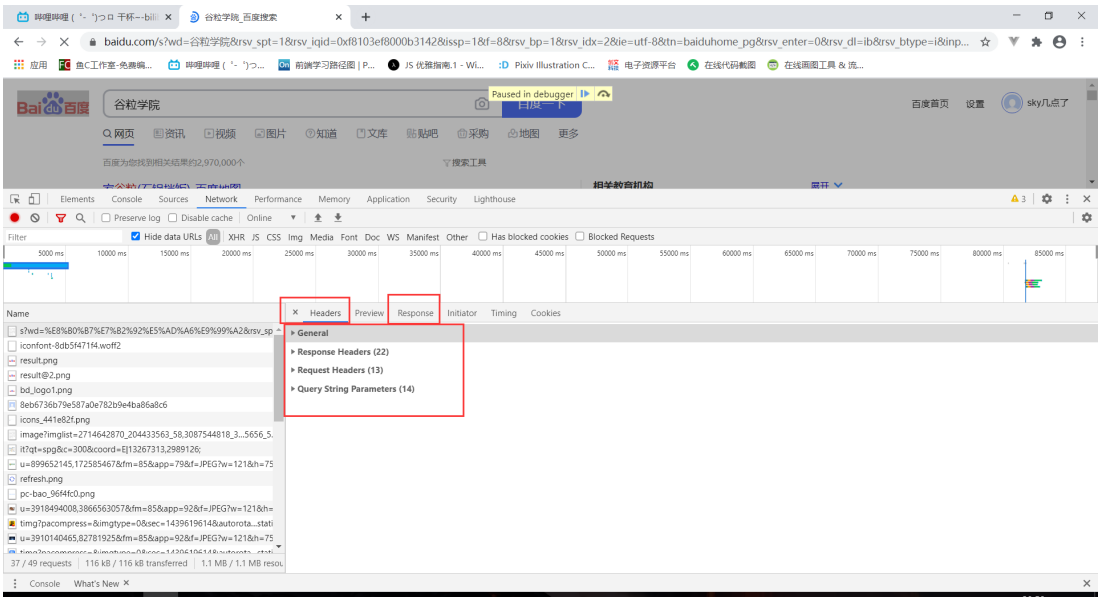
3. 响应空行

必须要有

4. 响应体

响应主要内容

3、get请求演示

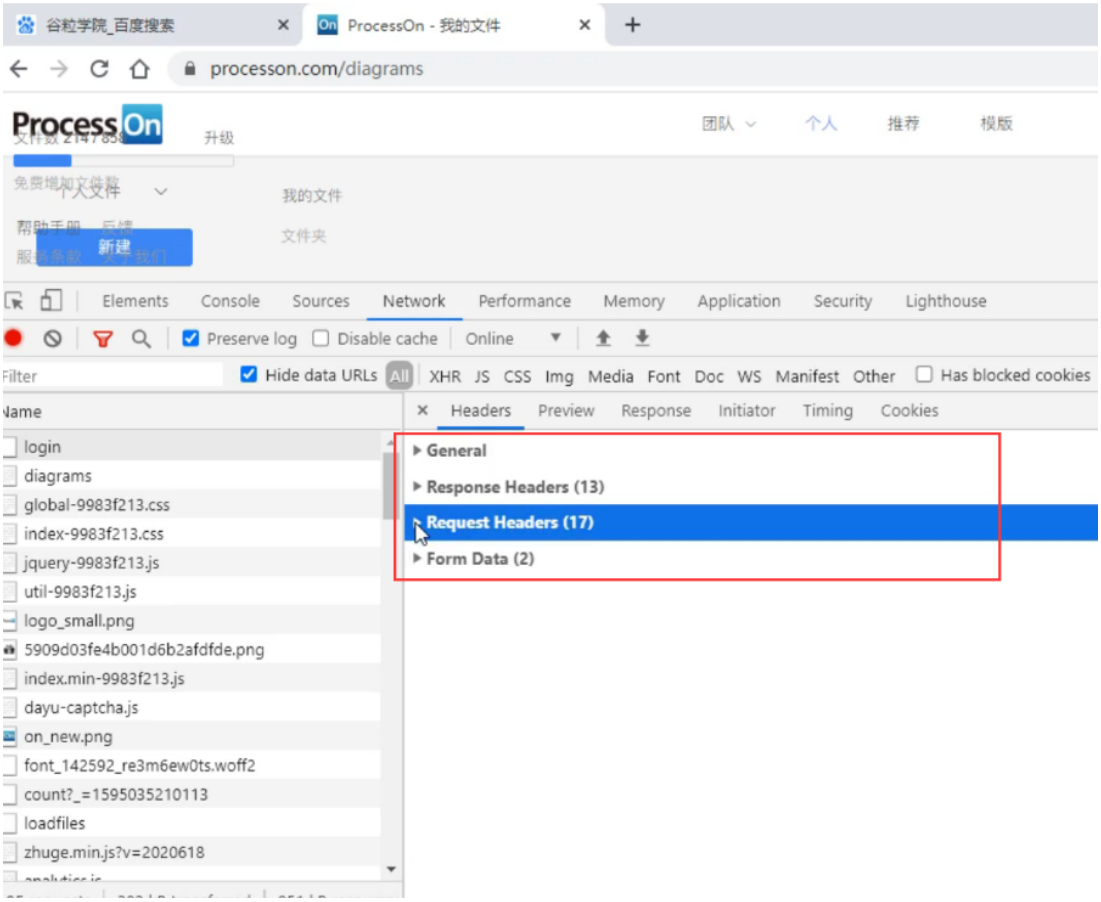


Headers：

- General
- Request Headers为请求头信息
- Response Headers为响应头信息
- Query String Parameters查询字符串参数:对url参数的解析

Response为服务端返回的html内容

4、post请求演示



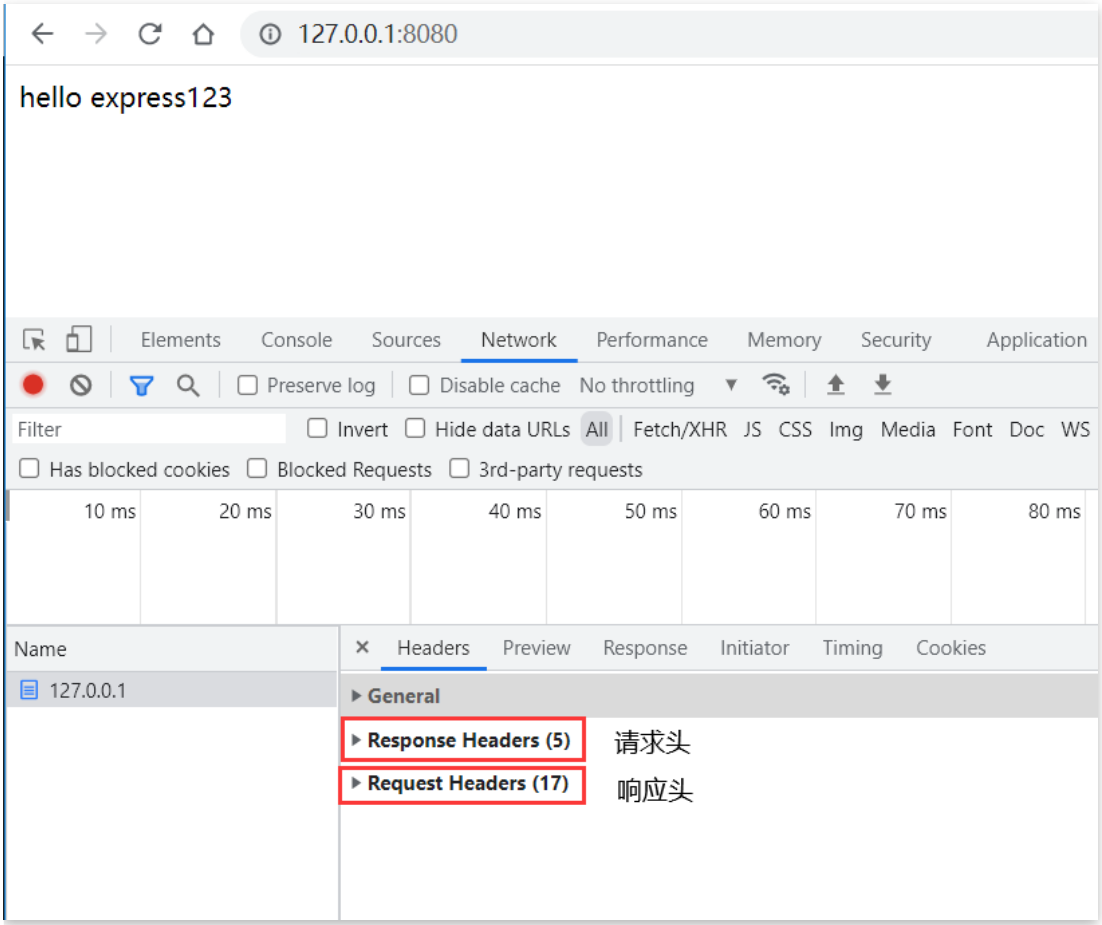
Form Data为请求体的内容

3、原生Ajax

1 express基本使用

```
//引入express并创建应用对象
const express = require('express')
const app = express()

//创建路由规则
app.get('/', (requesr, response)=>{
    //设置响应
    response.send('hello express')
})

//监听端口启动服务
app.listen(8000, ()=>{
    console.log('服务已经启动，端口8000监听中')
})
```

通过node.js启动后可以在127.0.0.1:8080访问



2 ajax请求基本步骤

get请求

html

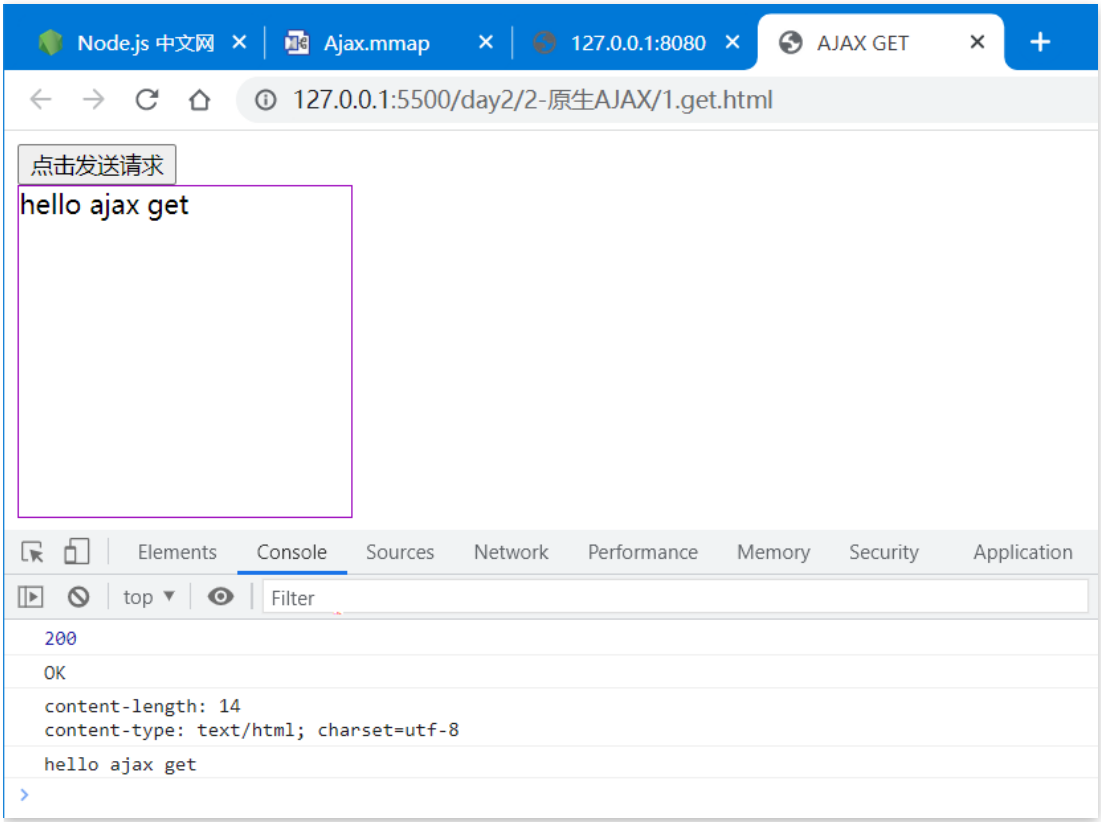
```html
<!DOCTYPE html>
<html lang="en">
<head>...
</head>
<body>
    <!-- 需求: 点击发送请求，将相应信息返回显示至ajax中 页面不刷新 -->
    <button>点击发送请求</button>
    <div id="res"></div>
    <script>
        const btn = document.getElementsByTagName('button')
        const result = document.getElementById('res')
        btn[0].onclick = function (){
            //创建对象
            const xhr = new XMLHttpRequest()
            //设置请求方法和url
            xhr.open('GET', 'http://localhost:8000/server')
            //发送
            xhr.send()
            //事件绑定 处理服务端返回的结果
            xhr.onreadystatechange = function (){
                if(xhr.readyState === 4){ //服务端返回了所有结果
                     //2开头的都表示成功
                    if(xhr.status >= 200 && xhr.readyState<=300){
                        //处理结果
                        console.log(xhr.status) //状态码
                        console.log(xhr.statusText) //状态字符串
                        console.log(xhr.getAllResponseHeaders) //所有响应头
                        console.log(xhr.response) //响应体
                        result.innerHTML = xhr.response
                    }else{}
                }
            }
        }
    </script>
</body>
</html>
```

```html
<button>点击发送请求</button>
    <div id="result"></div>
    <script>
        //1
        const btn = document.getElementsByTagName('button')[0];
        const result = document.getElementById('result');
        //2
        btn.onclick = function () {
            //1 创建XMLHttpRequest对象
            const xhr = new XMLHttpRequest();
            //2 初始化：用何种方式发请求？给谁发请求？带着什么参数过去？
```

```
            xhr.open('GET', 'http://localhost:8000/server?a=100&b=200&c=300');
            //3 发送
            xhr.send();
            //4 事件绑定
            //on when 当...时候
            //xhr内部有5种状态，标识着当前xhr内部读取数据的状态,分别是:
            //0: 开头 表示未初始化
            //1: 开头 表示open方法调用完毕
            //2: 开头 表示send方法调用完毕
            //3: 开头 表示服务端返回部分结果
            //4: 开头 表示服务端返回所有结果

            //5 处理服务端返回结果
            xhr.onreadystatechange = function () {
                //判断（服务端返回了所有结果）
                if (xhr.readyState === 4) {
                    //判断状态码200 404 403 401 500
                    //2xx 成功  2开头的都表示成功
                    if (xhr.status >= 200 && xhr.status < 300) {
                        //处理结果 行 头 空行 体
                        //响应行
                        console.log(xhr.status); //状态码
                        console.log(xhr.statusText); //状态字符串
                        console.log(xhr.getAllResponseHeaders()); //所有响应头
                        console.log(xhr.response); //响应体
                        //设置 result 文本
                        result.innerHTML = xhr.response;
                    } else {}
                }
            }
        }
    </script>
```

服务端:

```
//引入express并创建应用对象
const express = require('express')
const app = express()

//创建路由规则
app.get('/server', (requesr, response)=>{
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    //设置响应
    response.send('hello ajax')
})

//监听端口启动服务
app.listen(8000, ()=>{
    console.log('服务已经启动，端口8000监听中')
})

```



3、post请求

代码演示：

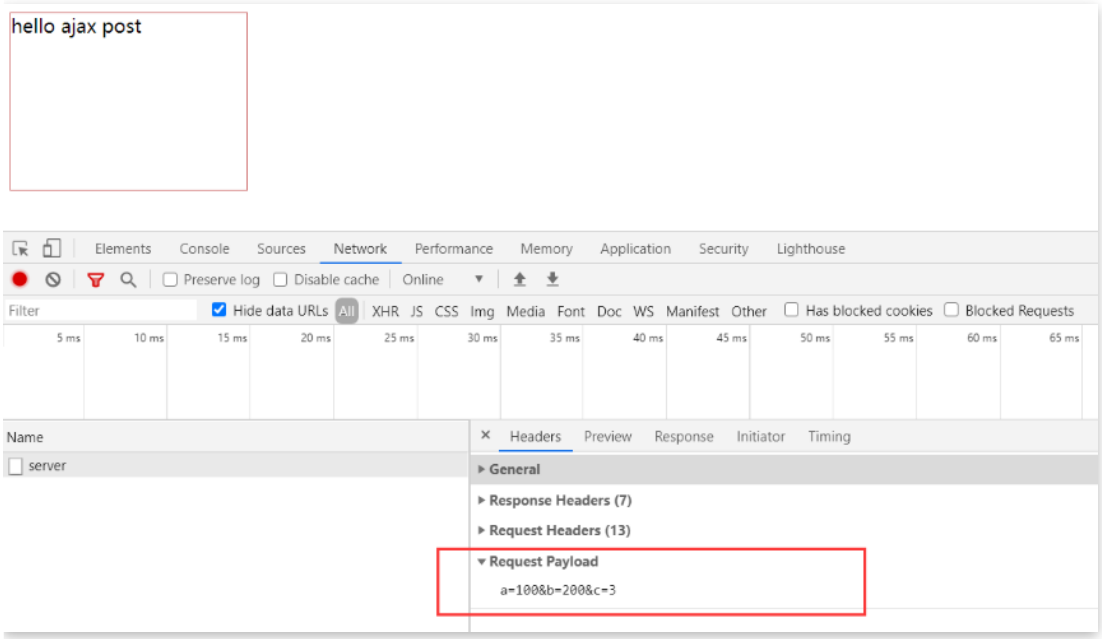
```html
<!DOCTYPE html>
<html lang="en">
<head>...
</head>
<body>
    <div id="res"></div>
    <script>
        const result = document.getElementById('res')
        result.onmouseover = function(){
            const xhr = new XMLHttpRequest()
            //与上一个GET请求相比，只变化了这一部分
            xhr.open('POST', 'http://localhost:8000/server')
            xhr.send()
            xhr.onreadystatechange = function(){
                if(xhr.readyState === 4){
                    if(xhr.status >= 200 && xhr.status<300){
                        result.innerHTML = xhr.response
                    }
                }
            }
        }
    </script>
</body>
</html>
```

```node
//引入express并创建应用对象
const express = require('express')
const app = express()

//创建路由规则
app.get('/server', (requesr, response)=>{
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    //设置响应
    response.send('hello ajax')
})
//接受post请求
app.post('/server', (requesr, response)=>{
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    //设置响应
    response.send('hello ajax post')
})

//监听端口启动服务
app.listen(8000, ()=>{
    console.log('服务已经启动，端口8000监听中')
})
```

post请求体参数设置：

位置：设置在 xhr.send()内

格式：'a=100&b=200&c=300' 或 'a:100&b:200&c:2'



设置请求头:

- 专用方法：xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
- 参数：接收两个参数，头的名字和头的值 同样可以在network中查看

```
<body>
    <div id="result"></div>
    <script>
        //1
        const result = document.querySelector('#result');
        //2
        result.addEventListener('click', function () {
            //console.log('test');
            const xhr = new XMLHttpRequest();
            xhr.open('post', 'http://127.0.0.1:8000/server');

            //设置请求头
            xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
            xhr.setRequestHeader('name', 'atguigu');

            xhr.send('a=100&b=200&c=300');
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4) {
                    if (xhr.status >= 200 && xhr.status < 300) {
                        result.innerHTML = xhr.response;
                    }
                }
```

```
            }
        })
    </script>
</body>
```

4、服务端响应json数据

```
<!DOCTYPE html>
<html lang="en">
<head>...
</head>
<body>
    <div id="res"></div>
    <script>
        const result = document.getElementById('res')
        window.onkeydown = function(){
            const xhr = new XMLHttpRequest()
            //设置响应体数据的类型
            xhr.responseType = 'json'
            xhr.open('GET', 'http://localhost:8000/json-server')
            xhr.send()
            xhr.onreadystatechange = function(){
                if(xhr.readyState === 4){
                    if(xhr.status>=200 && xhr.status<300){
                        console.log(xhr.response)

                        //手动对数据进行转换
                        // let data = JSON.parse(xhr.response)
                        // console.log(data)

                        //自动转换 上部须设置响应体数据类型
                        console.log(xhr.response)
                        result.innerHTML = xhr.response.name
                    }
                }
            }
        }
    </script>
</body>
</html>
```

```
//引入express并创建应用对象
const express = require('express')
const app = express()

//创建路由规则
app.get('/server', (requesr, response)=>{...
})
app.post('/server', (requesr, response)=>{...
})

//创建路由规则
app.get('/json-server', (requesr, response)=>{
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    //相应一个数据
    const data = {
        name:'hanser'
    }
    //对对象进行字符串转化
    let str = JSON.stringify(data)
    //设置响应
    response.send(str)
})
//监听端口启动服务
app.listen(8000, ()=>{
    console.log('服务已经启动，端口8000监听中')
})

```

5、Ajax请求超时与网络异常处理

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX 超时处理</title>
    <style>
        #result {
            width: 200px;
            height: 200px;
            border: 1px solid pink;
        }
    </style>
```

```
</head>

<body>
    <button>点击</button>
    <div id="result"></div>
    <script>
        const btn = document.querySelector('button');
        const result = document.querySelector('#result');

        result.addEventListener('click', function () {
            const xhr = new XMLHttpRequest();
            //超时设置
            xhr.timeout = 2000;
            //超时回调
            xhr.ontimeout = function () {
                alert('网络异常，请稍后再试')
            }
            //网络异常回调
            xhr.onerror = function () {
                alert('你的网络似乎出了问题')
            }

            xhr.open('get', 'http://127.0.0.1:8000/delay');
            xhr.send('');
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4) {
                    if (xhr.status >= 200 && xhr.status < 300) {
                        result.innerHTML = xhr.response;
                    }
                }
            }
        })
    </script>
</body>

</html>
```

6、取消请求

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX 取消请求</title>
</head>

<body>
    <button>点击发送</button>
    <button>点击取消</button>
    <script>
        const btns = document.querySelectorAll('button')
        let xhr = null;
        btns[0].onclick = function(){
            xhr = new XMLHttpRequest()
            xhr.open('GET', 'http://localhost:8000/delay')
            xhr.send();
        }
        //取消请求的回调函数 注意需要将xhr放在外侧
        btns[1].onclick = function(){
            xhr.abort();
        }
    </script>
</body>

</html>
```

7、解决请求重复发送的问题

如果连续发送同样的请求，会给服务器造成很大的不必要的压力。

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
```

```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX 重复请求</title>
</head>

<body>
    <button>点击发送</button>
    <script>
        const btns = document.querySelectorAll('button');
        let xhr = null;
        //标识变量
        let flag = false;
        btns[0].onclick = function () {
            //判断标识变量
            if (flag) xhr.abort();
            xhr = new XMLHttpRequest();
            //修改标识变量的值
            flag = true;
            xhr.open('GET', 'http://localhost:8000/delay');
            xhr.send();
            xhr.onreadystatechange = function () {
                //readyState=4：响应的内容解析完毕，可以在客户端使用了--完成
                if (xhr.readyState === 4) {
                    //改变标识变量
                    flag = false;
                }
            }
        }
    </script>
</body>

</html>
```

server:

```
//1 引入express
const express = require('express');

//2 创建应用对象
```

```
const app = express();

//3 创建路由规则
//get请求
app.get('/server', (request, response) => {
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*');
    //设置响应体
    response.send('hello ajax get');
});

//post请求
app.post('/server', (requesr, response) => {
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    //设置响应
    response.send('hello ajax post')
});

//JSON响应
app.all('/json-server', (request, response) => {
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*');
    //响应头
    response.setHeader('Access-Control-Allow-Headers', '*');
    //响应一个数据
    const data = {
        name: 'guang'
    };
    //对对象进行字符串转化
    let str = JSON.stringify(data);
    //设置响应体
    response.send(str);
});

//延时响应
app.all('/delay', (request, response) => {
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*');
    setTimeout(() => {
```

```
        //设置响应体
        response.send('延时响应');
    }, 3000)
});

//4 监听端口启动服务
app.listen(8000, () => {
    console.log('服务器启动了, 8000端口监听中...');
});
```

4、axios

1 基本使用

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>axios发送ajax请求</title>
    <style>
        button {
            border-radius: 5px;
            height: 35px;
            width: 100px;
            background-color: blueviolet;
            float: left;
            margin-right: 10px;
            color: white;
            border: none;
            text-align: center;
        }

        .btn-primary {
            margin-left: 160px;
            background-color: #cd6969;
        }

```

```
        .btn-danger {
            background-color: orange;
        }
    </style>
    <script src="https://cdn.bootcdn.net/ajax/libs/axios/0.19.2/axios.js"></script>
</head>

<body>
    <button class="btn btn-primary">GET</button>
    <button class="btn btn-danger">POST</button>
    <button class="btn btn-info">通用型方法</button>
    <script>
        const btns = document.querySelectorAll('button')
        //1 get请求
        //第一个参数为url地址，第二个为其他参数配置的对象
        btns[0].onclick = function () {
            axios.get('http://localhost:8000/axios-server', {
                //url参数
                params: {
                    id: 1000,
                    vip: 12
                },
                //请求头信息
                headers: {
                    name: 'hanser',
                    age: '2'
                }
            }).then(value => {
                console.log(value)
            })
        }
        //2 post请求
        //第一个参数为url地址，第二个为请求体，第三个为其他参数配置
        btns[1].onclick = function () {
            axios.post('http://localhost:8000/axios-server', {
                username: 'admin',
                password: '123'
            }, {
                //url参数
                params: {
```

```
                    id: 1,
                    vip: 123
                },
                //请求头参数
                headers: {
                    name: 'yousa',
                    age: '23'
                },
                //请求体
            })
        }
        //3 通用方式发送
        //参数只有一个对象，参数顺序与报文顺序一致
        btns[2].onclick = function () {
            axios({
                method: 'POST',
                url: 'http://localhost:8000/axios-server',
                //url参数
                params: {
                    vip: 10,
                    id: 123
                },
                //头信息
                headers: {
                    a: 100,
                    b: 200
                },
                //请求体参数
                data: {
                    name: 'hanser',
                    age: '7'
                }
            }).then(response => {
                console.log(response)
            })
        }
    </script>
</body>
</html>
```

server:

```
//1 引入express
const express = require('express');

//2 创建应用对象
const app = express();

//3 创建路由规则
// axios 服务
app.all('/axios-server', (request, response) => {
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    response.setHeader('Access-Control-Allow-Headers', '*')
    const data = { name: 'hanser' }
    //设置响应
    response.send(JSON.stringify(data))
})

//4 监听端口启动服务
app.listen(8000, () => {
    console.log('服务器启动了, 8000端口监听中...');
});
```

2、fetch函数发送ajax请求

```html
<script>
    const btn = document.getElementsByTagName('button')
    btn[0].onclick = function(){
        fetch('http://localhost:8000/fetch-server',{
            //请求方法
            method:'POST',
            //请求头
            headers:{
                name:'hanser'
            },
            //请求体
            body: 'name=admin&pd=admin'
        }).then(Response=>{
            console.log(Response)
            return Response.text()
        }).then(Response=>{
            console.log(Response)
        })
    }
</script>
```

server:

```js
//1 引入express
const express = require('express');

//2 创建应用对象
const app = express();

//3 创建路由规则
// fetch 服务
app.all('/fetch-server', (requesr, response) => {
    //设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    response.setHeader('Access-Control-Allow-Headers', '*')

    const data = { name: 'hanser' }
    //设置响应
    response.send(JSON.stringify(data))
})

//4 监听端口启动服务
app.listen(8000, () => {
    console.log('服务器启动了, 8000端口监听中...');
```

```
});
```

5、跨域

1 Ajax同源策略

同源策略是浏览器的一种安全策略，违背同源策略就是跨域

同源:协议，域名，端口号必需完全相同

2 如何解决跨域

jsonp解决跨域问题：

ajax响应的返回结果是函数调用，而函数的实参就是我们想给客户端返回的结果数据，而这个函数必须于线定义好

思路就是往文件里添加script标签，标签引入内容的格式为上方所述

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>跨域</title>
</head>

<body>
    <h1>hanser</h1>
    <button>点击获取用户数据</button>
    <script>
        const btn = document.getElementsByTagName('button')
        btn[0].onclick = function () {
            const x = new XMLHttpRequest()
            //因为满足同源策略，所以url可以简写
            x.open('GET', '/data')
            x.send()
            x.onreadystatechange = function () {
                if (x.readyState === 4) {
                    if (x.status >= 200 && x.status < 300) {
                        console.log(x.response)
                    }
```

```
                }
            }
        }
    </script>
</body>

</html>
```

服务端：

```
const express = require('express')

const app = express()
app.get('/home', (request, response) => {
    response.sendFile(__dirname + '/index.html')
})

app.get('/data', (request, response) => {
    response.send('用户数据')
})

app.listen(9000, () => {
    console.log('端口9000服务已经启动')
})
```

6 取消请求

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>练习取消请求</title>
</head>
<body>

  <button id="btn">点我获取验证码</button>
  <div id="test"></div>

```

```
  <script type="text/javascript">
    let btn = document.getElementById('btn')
    let lastXhr
    btn.onclick = function () {
      if (lastXhr) lastXhr.abort()
      lastXhr = getCode()
    }

    function getCode() {
      let xhr = new XMLHttpRequest()
      xhr.onreadystatechange = function () {
        if (xhr.readyState === 4 && xhr.status === 200) {
          console.log('验证码是: ', xhr.response)
        }
      }
      xhr.open('get', 'http://localhost:3000/get_verify_code')
      xhr.send()
      return xhr
    }
  </script>

</body>
</html>
```

server:

```
//引入express
const express = require('express')

//实例一个app应用对象
const app = express()

//使用内置中间件去解析post请求中以urlencoded形式编码的参数
app.use(express.urlencoded({ extended: true }))

//暴露静态资源
app.use(express.static(__dirname + '/public'))

app.get('/get_verify_code', (req, res) => {
```

```
  console.log('有人找我要验证码了')
  setTimeout(() => {
    let code = Math.floor(Math.random() * 8999 + 1000)
    res.send(code.toString())
  }, 1000)
})

//
app.listen(3000, (err) => {
  if (err) console.log(err)
  else {
    console.log('不要用编译器打开页面，会产生跨域问题')
    console.log('地址为：http://localhost:3000/code.html')
  }
})
```

2 跨域

1.为什么会有跨域这个问题?

原因是浏览器为了安全，而采用的同源策略（Same origin policy）

2.什么是同源策略?

1. 同源策略是由Netscape提出的一个著名的安全策略，现在所有支持JavaScript 的浏览器都会使用这个策略。

2. Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。

3. 所谓同源是指：协议，域名（IP），端口必须要完全相同

即：协议、域名（IP）、端口都相同，才能算是在同一个域里。

3.非同源受到哪些限制?

1. Cookie不能读取;

2. DOM无法获得;

3. Ajax请求不能获取数据

4.如何在开发中解决跨域问题:

1.JSONP解决发送请求跨域问题:

要明确的是：JSONP不是一种技术，而是程序员“智慧的结晶”（利用了标签请求资源不受同源策略限制的特点）

JSONP需要前后端人员互相配合。

前端页面写法:

```
<body>
```

```
  <button id="btn">按钮</button>
  <script type="text/javascript">
    var btn = document.getElementById('btn');
    btn.onclick = function () {
      //1. 创建一个script标签
      var script = document.createElement('script');
      //2. 设置回调函数
      window.getData = function (data) {
        console.log(data);//拿到数据
      }
      //3. 设置script标签src属性，填写跨域请求的地址
      script.src = 'http://localhost:3000/jsonp?callback=getData';
      //4. 将script标签添加到body中生效
      document.body.appendChild(script);
      //5.不影响整体DOM结构，删除script标签
      document.body.removeChild(script);
    }
  </script>
</body>
```

后端写法：

```
app.get('/jsonp', (req, res) => {
  //解构赋值获取请求参数
  const { callback } = req.query
  //去数据库查找对应数据
  const data = [{ name: 'tom', age: 18 }, { name: 'jerry', age: 20 }];
  res.send(callback + '(' + JSON.stringify(data) + ')');
})
```

2.后台配置cors解决跨域**

以Node为例：

res.set('Access-Control-Allow-Origin', 'http://localhost:63342');

3.使用代理服务器

例如：nginx等

3. JSONP解决跨域

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Title</title>
   <script type="text/javascript" src="./jquery.min.js"></script>
</head>
<body>
  <button id="btn1">点我使用jquery封装的jsonp获取数据</button>

  <script type="text/javascript">
    let btn1 = $('#btn1')
    btn1.click(() => {
      //完整写法
      /*$.ajax({
        url:'http://localhost:3000/test_get',
        method:'get',
        dataType:'jsonp',
        data:{name:'kobe',age:18},
        success:(result)=>{
          console.log(result)
        },
        error:(err)=>{
          console.log(err)
        }
      })*/

      //简单写法
      $.getJSON('http://localhost:3000/test_get?callback=?', { name: 'kobe', age: 18 }, (data) => {
        console.log(data)
      })
    })
  </script>

</body>
</html>
```

```
<!DOCTYPE html>
```

```html
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>原生js写jsonp</title>
</head>

<body>
  <button id="btn">点我获取数据</button>
  <form action="http://localhost:3000/test_get" method="get">
    <input type="submit">
  </form>

  <script type="text/javascript">
    /*
    * jsonp解决跨域：
    *     1.利用script标签，发请求不受到同源策略的限制。
    *     2.需要后端工程师配合
    *     3.不可以解决post请求跨域的问题
    * */

    let btn = document.getElementById('btn')
    btn.onclick = () => {
      //1.创建一个script标签
      let scriptNode = document.createElement('script')
      //2.提前定义好一个函数
      window.demo3 = function (data) {
        console.log(data)
      }
      //3.为script标签指定src属性
      scriptNode.src = 'http://localhost:3000/test_get?callback=demo3'
      //4.将script标签加入页面
      document.body.appendChild(scriptNode)
    }
  </script>

</body>
</html>
```

```
const express = require('express')

const app = express()

app.use(express.urlencoded({ extended: true }))

app.use(express.json())

app.use(express.static(__dirname + '/public'))

app.get('/test_get', (req, res) => {
  const { callback } = req.query
  console.log(callback)
  let person = [{ name: 'kobe', age: 18 }, { name: 'kobe', age: 18 }, { name: 'kobe', age: 18 }, { name: 'kobe', age: 18 }]
  res.send(`${callback}(${JSON.stringify(person)})`)
})

app.listen(3000, (err) => {
  if (err) console.log(err)
  else {
    console.log('兄弟必须要用编译器打开页面，你要解决跨域问题')
  }
})
```

4.jQuery封装ajax：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>封装ajax</title>
</head>

<body>
  <button id="btn">点我使用自己封装的ajax请求数据</button>
  <script type="text/javascript">

```

```
let btn = document.getElementById('btn')
btn.onclick = () => {
  sendAjax({
    url: 'http://localhost:3000/test_post',
    method: 'post',
    data: { name: 'kobe', age: 18 },
    success: (result) => {
      console.log(result)
    },
    error: (err) => {
      console.log(err)
    }
  })
}

//用于发送ajax请求的函数，封装的是原生xhr
function sendAjax(options) {

  //1.从配置对象中获取发请求所需的参数
  let { url, method, data, success, error } = options
  method = method || 'get'

  //2.实例化一个xhr
  let xhr = new XMLHttpRequest()

  //3.监听状态
  xhr.onreadystatechange = () => {
    if (xhr.readyState !== 4) return
    if (xhr.readyState === 4 && (xhr.status >= 200 && xhr.status <= 299)) {
      if (success) success(xhr.response)
    } else {
      if (error) error('请求失败')
    }
  }

  //4.整理参数
  let str = ''
  if (data) {
    for (let key in data) {
```

```
            str += `${key}=${data[key]}&`
          }
        }

        //5.此处应该判断请求方式，根据请求的方式，决定如何携带参数
        if (method.toUpperCase() === 'GET') {
          xhr.open(method, `${url}?${str}`)
          xhr.send()
        } else {
          xhr.open(method, url)
          xhr.setRequestHeader('content-type', 'application/x-www-form-urlencoded')
          xhr.send(str)
        }

      }
    </script>

</body>

</html>
```

```
//引入express
const express = require('express')
//实例一个app应用对象
const app = express()
//使用内置中间件去解析post请求中以urlencoded形式编码的参数
app.use(express.urlencoded({ extended: true }))
app.use(express.json())
//暴露静态资源
app.use(express.static(__dirname + '/public'))

app.get('/test_get', (req, res) => {
  res.set('Access-Control-Allow-Origin', '*')
  console.log('有人发来了get请求')
  console.log(req.query)
  res.send('你发来的是get请求，我收到了，这是给你的数据：哈哈')
})

```

```
app.post('/test_post', (req, res) => {
  res.set('Access-Control-Allow-Origin', '*')
  console.log(req.body);
  res.send('你发来的是post请求，我收到了，这是给你的数据：哈哈')
})

app.listen(3000, (err) => {
  if (err) console.log(err)
  else {
    console.log('服务器启动成功')
  }
})
```